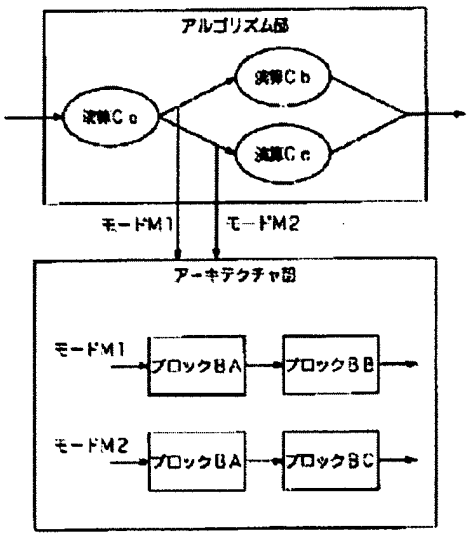
METHOD AND DEVICE FOR EVALUATING PERFORMANCE

Publication number: JP2000311186
Publication date: 2000-11-07
Inventor: HASHIMOTO TAKEHISA; ISHIKAWA HITOSHI
Applicant: SONY CORP
Classification:
- international: ***G06F17/50;*** **G06F17/50;** (IPC1-7): G06F17/50
- European:
Application number: JP19990122733 19990428
Priority number(s): JP19990122733 19990428

Report a data error here

Abstract of **JP2000311186**

PROBLEM TO BE SOLVED: To shorten man-hour for design for performance evaluation and to more shorten time for simulation. SOLUTION: A description concerning a performance evaluation model for evaluating the performance of the hardware structure of an LSI is separated to an algorithm part expressing the function of the hardware structure and an architecture part to affect the performance of the hardware structure and the performance of the hardware structure is evaluated on the basis of the description in the algorithm part and the description in the architecture part. Besides, a mode M is set as an interface for relating the separated algorithm part and architecture part.



(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2000－311186
（Ｐ2000－311186Ａ）
(43)公開日　平成12年11月７日(2000.11.7)

(51)Int.Cl.⁷	識別記号	ＦＩ	ﾃｰﾏｺｰﾄﾞ(参考)
Ｇ０６Ｆ　17/50		Ｃ０６Ｆ　15/60	６６４Ｋ　５Ｂ０４６

審査請求　未請求　請求項の数33　ＯＬ　（全 15 頁）

(21)出願番号　特願平11－122733

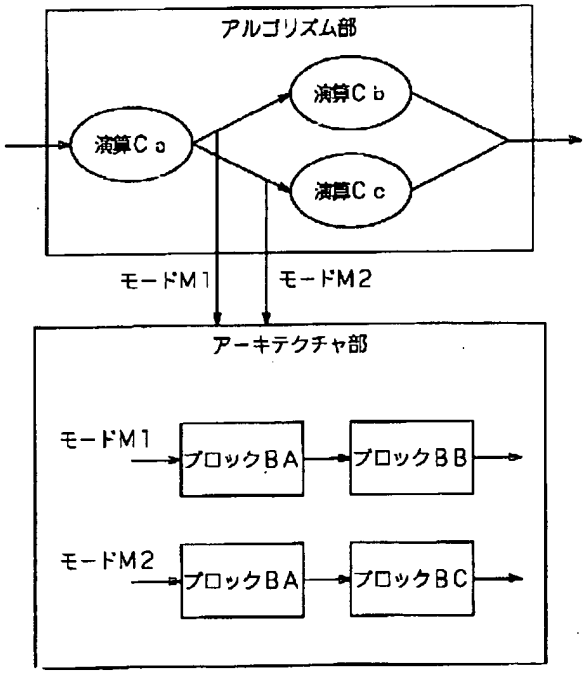
(22)出願日　平成11年４月28日(1999.4.28)

(71)出願人　000002185
ソニー株式会社
東京都品川区北品川６丁目７番35号
(72)発明者　橋本　毅久
東京都品川区北品川６丁目７番35号　ソニー株式会社内
(72)発明者　石川　仁
東京都品川区東五反田１丁目14番10号　株式会社ソニー木原研究所内
(74)代理人　100067736
弁理士　小池　晃　（外２名）

Ｆターム(参考)　５Ｂ046 AA08 BA02 JA04

(54)【発明の名称】　性能評価方法及び装置

(57)【要約】
【課題】　性能評価のための設計工数を短くし、シミュレーションの時間をより短縮可能とする。
【解決手段】　ＬＳＩのハードウェア構造の性能を評価するための性能評価モデルに関する記述を、ハードウェア構造の機能を表現しているアルゴリズム部分とハードウェア構造の性能に影響を及ぼすアーキテクチャ部分とに分離し、アルゴリズム部分の記述とアーキテクチャ部分の記述とに基づいて、ハードウェア構造の性能を評価する。また、分離されているアルゴリズム部分とアーキテクチャ部分とを関連付けるインターフェイスとしてモードＭを設定している。



本発明の性能評価モデルの構造

【特許請求の範囲】

【請求項１】　ハードウェア構造を計算機により演算可能な記述としてモデル化し、当該モデル化したハードウェア構造の性能を当該計算機を用いて評価する性能評価方法において、

上記ハードウェア構造の性能を評価するための性能評価モデルに関する記述を、上記ハードウェア構造の機能を表現しているアルゴリズム部分と上記ハードウェア構造の性能に影響を及ぼすアーキテクチャ部分とに分離し、上記アルゴリズム部分の記述とアーキテクチャ部分の記述とに基づいて、上記ハードウェア構造の性能を評価することを特徴とする性能評価方法。

【請求項２】　上記分離されているアルゴリズム部分とアーキテクチャ部分とを関連付けるインターフェイスとしてモードを設定することを特徴とする請求項１記載の性能評価方法。

【請求項３】　上記アルゴリズム部分が行う機能的な演算によって上記性能が変化するとき、上記アルゴリズム部分は、上記性能の変化を異なるモードとして計算し、当該モードを上記アーキテクチャ部分に与えることを特徴とする請求項２記載の性能評価方法。

【請求項４】　上記アーキテクチャ部分は、上記ハードウェア構造のうちの性能に影響する部分を、データの入出力が可能なリソースの結合されたものとして保持し、それらリソースの間のデータの入出力を制御することにより、上記性能を求めることを特徴とする請求項２記載の性能評価方法。

【請求項５】　上記リソース間でデータ入出力を行う制御のための記述を、上記リソースの種類に依存しない性質に関する記述と、上記リソースの種類に依存する性質に関する記述とに分離して保持することを特徴とする請求項４記載の性能評価方法。

【請求項６】　上記リソースの種類に依存しない性質に関する記述は、任意のリソースを最小単位のリソースの結合とする旨の記述を含むことを特徴とする請求項５記載の性能評価方法。

【請求項７】　上記リソースの結合とは、単一のリソースから複数のリソースに結合する分岐と、複数のリソースから単一のリソースに結合する集約とを含むことを特徴とする請求項６記載の性能評価方法。

【請求項８】　上記リソースは、上記リソースの結合を上記モードに応じて複数個所持するリソースを含むことを特徴とする請求項６記載の性能評価方法。

【請求項９】　上記リソースの種類に依存しない性質に関する記述は、最小単位のリソース内に同時に２つ以上のデータの存在を許可しない旨の記述を含むことを特徴とする請求項５記載の性能評価方法。

【請求項１０】　上記リソースの種類に依存しない性質に関する記述は、任意のリソースはデータが出力可能であり、且つ上記リソースの結合により結合された先のリソースがデータを入力可能であるならば、データを必ず移動する旨の記述を含むことを特徴とする請求項６記載の性能評価方法。

【請求項１１】　上記リソースの種類に依存する性質に関する記述は、上記リソースのデータ入力が可能かどうかの判断において、データを保持しているか否か、及び上記リソースの結合において後方に結合されたリソースがデータ入力可能か否か、のいずれの情報を利用するかを表す記述を含むことを特徴とする請求項６記載の性能評価方法。

【請求項１２】　上記リソースの種類に依存する性質に関する記述は、上記リソースの結合が上記分岐である時に、どのような情報を利用するかを表す記述を含むことを特徴とする請求項７記載の性能評価方法。

【請求項１３】上記リソースの種類に依存する性質に関する記述は、上記リソースの結合が上記集約である時に、どのような情報を利用するかを表す記述を含むことを特徴とする請求項７記載の性能評価方法。

【請求項１４】　上記リソースの種類に依存しない性質及び上記リソースの種類に依存する性質に関する記述をライブラリ情報として提供し、

当該ライブラリ情報の組み合わせによってリソースを定義し、

それらリソースの結合を記述することにより、上記性能を評価することを特徴とする請求項５記載の性能評価方法。

【請求項１５】　上記アルゴリズム部分と上記アーキテクチャ部分の間で、複数個のモードを保持することを特徴とする請求項２記載の性能評価方法。

【請求項１６】　上記リソースにデータが入力されるとき、上記リソースが保持する上記モードに対応したリソースの結合の先頭リソースに対してデータを入力することを特徴とする請求項８記載の性能評価方法。

【請求項１７】　上記リソースが保持する複数のリソースの結合において、出力側に仮想的に１つのリソースを設けることを特徴とする請求項８記載の性能評価方法。

【請求項１８】　ハードウェア構造を計算機により演算可能な記述としてモデル化し、当該モデル化したハードウェア構造の性能を評価する性能評価装置において、

上記ハードウェア構造の性能を評価するための性能評価モデルに関する記述を、上記ハードウェア構造の機能を表現しているアルゴリズム部分と上記ハードウェア構造の性能に影響を及ぼすアーキテクチャ部分とに分離した記述を保持する保持手段と、

上記保持手段に保持したアルゴリズム部分の記述に基づいて演算を行うアルゴリズム演算手段と、

上記保持手段に保持したアーキテクチャ部分の記述に基づいて演算を行うアーキテクチャ演算部とを有し、

上記アルゴリズム演算手段とアーキテクチャ演算手段の演算により、上記ハードウェア構造の性能を評価するこ

とを特徴とする性能評価装置。

【請求項１９】　上記アルゴリズム演算手段は、入力されたデータに対する演算結果にしたがって、上記分離されているアルゴリズム部分とアーキテクチャ部分とを関連付けるインターフェイスとしてのモードを計算するモード計算手段を有することを特徴とする請求項１８記載の性能評価装置。

【請求項２０】　上記モード計算手段は、上記アルゴリズム部分が行う機能的な演算によって上記性能が変化するとき、上記アルゴリズム部分の記述に基づいて上記性能の変化を異なるモードとして計算し、当該モードを上記アーキテクチャ演算手段に送出することを特徴とする請求項１９記載の性能評価装置。

【請求項２１】　上記アーキテクチャ演算手段は、上記ハードウェア構造のうちの性能に影響する部分を、データの入出力が可能なリソースの結合されたものとして保持するリソース保持手段と、それらリソースの間のデータの入出力を制御するデータ制御手段とを有することを特徴とする請求項１９記載の性能評価装置。

【請求項２２】　上記保持手段には、上記リソース間でデータ入出力を行うための記述を、上記リソースの種類に依存しない性質に関する記述と、リソースの種類に依存する性質に関する記述とに分離して保持することを特徴とする請求項２１記載の性能評価装置。

【請求項２３】　上記リソースの種類に依存しない性質に関する記述は、任意のリソースを最小単位のリソースの結合とする旨の記述を含むことを特徴とする請求項２２記載の性能評価装置。

【請求項２４】　上記リソースの結合とは、単一のリソースから複数のリソースに結合する分岐と、複数のリソースから単一のリソースに結合する集約とを含むことを特徴とする請求項２３記載の性能評価装置。

【請求項２５】　上記リソースは、上記リソースの結合を上記モードに応じて複数個所持するリソースを含むことを特徴とする請求項２３記載の性能評価装置。

【請求項２６】　上記リソースの種類に依存しない性質に関する記述は、最小単位のリソース内に同時に２つ以上のデータの存在を許可しない旨の記述を含むことを特徴とする請求項２２記載の性能評価装置。

【請求項２７】　上記リソースの種類に依存しない性質に関する記述は、任意のリソースはデータが出力可能であり、且つ上記リソースの結合により結合された先のリソースがデータを入力可能であるならば、データを必ず移動する旨の記述を含むことを特徴とする請求項２３記載の性能評価装置。

【請求項２８】　上記リソースの種類に依存する性質に関する記述は、上記リソースのデータ入力が可能かどうかの判断において、データを保持しているか否か、及び上記リソースの結合において後方に結合されたリソースがデータ入力可能か否か、のいずれの情報を利用するかを表す記述を含むことを特徴とする請求項２３記載の性能評価装置。

【請求項２９】　上記リソースの種類に依存する性質に関する記述は、上記リソースの結合が上記分岐である時に、どのような情報を利用するかを表す記述を含むことを特徴とする請求項２４記載の性能評価装置。

【請求項３０】上記リソースの種類に依存する性質に関する記述は、上記リソースの結合が上記集約である時に、どのような情報を利用するかを表す記述を含むことを特徴とする請求項２４記載の性能評価装置。

【請求項３１】　上記アルゴリズム部分と上記アーキテクチャ部分の間で、複数個のモードを保持するモード保持手段を有することを特徴とする請求項１９記載の性能評価装置。

【請求項３２】　上記データ制御手段は、上記リソースにデータが入力されるとき、上記リソースが保持する上記モードに対応したリソースの結合の先頭リソースに対してデータを入力することを特徴とする請求項２５記載の性能評価装置。

【請求項３３】　上記データ制御手段は、リソースが保持する複数のリソースの結合において、出力側に仮想的に１つのリソースを設けることを特徴とする請求項２５記載の性能評価装置。

【発明の詳細な説明】

【０００１】

【発明の属する技術分野】本発明は、大規模集積回路（ＬＳＩ）のアーキテクチャ設計の性能評価方法及び装置に関する。

【０００２】

【従来の技術】ＬＳＩ設計者がＬＳＩを設計する場合の従来の設計フローを図２０に示す。

【０００３】従来の設計フローでは、最初に、ステップＳ１０１として、要求仕様に基づいて機能の設計を行い、次に、ステップＳ１０２として、評価のために、シミュレータとしての機能評価モデルを作成し、その後、ステップＳ１０３として、機能評価モデルが要求仕様を満たすかどうかを確認する。当該ステップＳ１０３において、もし機能評価モデルが要求仕様を満たすものでなければ（ＮＯ）、ステップＳ１０１に戻って再度機能を見直す。

【０００４】ステップＳ１０３において、機能評価モデルが要求仕様を満たしていることが確認できた（ＹＥＳ）後は、ステップＳ１０４として、機能の実現方法としてのアーキテクチャの設計を行う。ここで、アーキテクチャによって性能が左右されるので、ステップＳ１０５として、実行時間をシミュレートするための性能評価モデルを作成して、次に、ステップＳ１０６として、要求仕様を満たすがどうか確認する。このステップＳ１０６において、要求仕様を満たさない（ＮＯ）ものであれば、ステップＳ１０４に戻ってアーキテクチャの再設計

を行い、要求仕様を満たした（ＹＥＳ）ならば設計が完了する。

【０００５】ここで、機能評価モデルは、機能を数学的に表現したアルゴリズムと、アルゴリズムで処理するデータの入力又はその処理結果データの出力の何れか、或いはそれらの両方を持つ。なお、このレベルではハードウェアの処理にかかる時間は意識しない。機能評価モデルを作成する場合は、アルゴリズムをＣ言語として知られるような適切な高レベル言語で記述する。機能評価モデルは、要求仕様の中の機能が実現されたモデルであり、設計対象の論理的な振る舞いを表現しているという意味でビヘイビアモデルとも呼ばれる。

【０００６】一方、性能評価モデルでは、機能評価モデルで記述したアルゴリズムについて、実際のハードウェアでの処理にかかる時間を記述し、シミュレーションする。ただし、時間の記述には、２つのレベルが存在する。

【０００７】一つは、ハードウェアの１クロックを最小単位とし、回路を構成する論理ゲートや配線による遅延を考慮しない時間的な概念であり、一般にレジスタ転送レベル（ＲＴＬ）と呼ばれる。このような時間概念を、以下、レベル１の時間概念と呼ぶことにする。

【０００８】またもう一つは、ハードウェアのクロックの単位よりも詳細に、回路を構成する論理ゲートや配線による遅延も考慮に入れた時間的な概念である。このような時間概念を、以下、レベル２の時間概念と呼ぶことにする。

【０００９】このような時間概念の考慮された性能評価モデルのシミュレーションを行う公知の技術では、ハードウェア記述言語（ＨＤＬ）によって記述されたモデルを検証するＨＤＬシミュレータがある。 ＨＤＬシミュレータには、レベル１の時間概念に基づいてタイミングと機能を検証するサイクルベースシミュレータと、レベル２の時間概念に基づいて厳密な処理のタイミングと機能を検証するイベントドリブンシミュレータが存在する。

【００１０】また、別の性能評価モデルを作成して評価するための既存の技術として、レジスタ転送Ｃ（ＲＴＣ）言語による記述がある。これは既存のＣ言語にハードウェアのクロックの概念を持ち込み、１クロック毎の処理をＣ言語で記述することにより、レベル１の時間概念に基づいてタイミングと機能の検証を可能とするものである。

【００１１】ここで、レベル１の時間概念における、ＨＤＬによる性能評価モデルとＲＴＣによる性能評価モデルを比較してみる。

【００１２】ＨＤＬによる性能評価モデルによるシミュレーション実行の前提条件としてＨＤＬ言語による記述が必要である。 ＨＤＬ言語による記述は、ハードウェアクロックを意識した並行的な動作の記述という点においてＣ言語の記述とは異なる。したがって、機能評価モデルとして作成したものとは別途作成する必要がある。そのため、機能評価が完了した後に性能評価モデルを作成して性能評価を行うまでに設計工数が大きくかかるという問題がある。

【００１３】それに対して、ＲＴＣを利用する場合は、機能評価モデルをＣ言語で作成してあれば、それを流用して性能評価モデルを作成することが可能である。したがって、ＨＤＬシミュレータと比較して、性能評価を行うまでの設計工数が短くなる。

【００１４】また、ＨＤＬシミュレータは、前述の通り、ハードウェアを意識した詳細なレベルのシミュレーションを行うため、性能評価としての精度は良くなるが、レベル１での時間概念のレベルで必要となる抽象的な記述には向いていないため、モデルの抽象度が下がり、シミュレーションそのものにかかる時間も非常に長くなるという問題がある。それに対してＲＴＣを利用する場合は、Ｃ言語のレベルで動作するので、同等な時間概念のレベルであるＨＤＬのシミュレータよりもシミュレーションそのものの実行時間を短縮することができる。

【００１５】以上のことから、レベル１の時間概念のレベルでの性能評価には、ＨＤＬよりもＲＴＣを利用した性能評価モデルを用いた方が、性能評価を行うまでの設計工数及び性能評価のためのシミュレーション時間は短縮されることになる。

【００１６】

【発明が解決しようとする課題】しかし、ＲＴＣを用いた性能評価モデルにおいても、性能評価のための設計工数や、シミュレーション実行時間の短縮は未だ不十分である。

【００１７】すなわち機能評価モデルは、アルゴリズムの記述である演算のみの組み合わせから記述を行う。これは、ハードウェア構造に無関係である。したがって、ＲＴＣを用いた性能評価モデルであっても、ハードウェアを意識した記述を行うならば、たとえ利用する言語が同じであっても、再度書き直す必要が生じる。

【００１８】このことが、性能評価のための設計工数に大きく影響を与える。また、この場合の記述では、記述するレベルがハードウェアに非常に近く、Ｃ言語を使用していながら、抽象的な記述を行うことが少ない。

【００１９】そこで、本発明はこのような状況に鑑みてなされたものであり、性能評価のための設計工数を短くでき、また、シミュレーションの時間をより短縮することが可能な、性能評価方法及び装置を提供することを目的とする。

【００２０】

【課題を解決するための手段】本発明の性能評価方法は、ハードウェア構造を計算機により演算可能な記述としてモデル化し、当該モデル化したハードウェア構造の

性能を当該計算機を用いて評価する性能評価方法であり、上記ハードウェア構造の性能を評価するための性能評価モデルに関する記述を、上記ハードウェア構造の機能を表現しているアルゴリズム部分と上記ハードウェア構造の性能に影響を及ぼすアーキテクチャ部分とに分離し、上記アルゴリズム部分の記述とアーキテクチャ部分の記述とに基づいて、上記ハードウェア構造の性能を評価することにより、上述した課題を解決する。

【００２１】また、本発明の性能評価装置は、ハードウェア構造を計算機により演算可能な記述としてモデル化し、当該モデル化したハードウェア構造の性能を評価する性能評価装置であり、上記ハードウェア構造の性能を評価するための性能評価モデルに関する記述を、上記ハードウェア構造の機能を表現しているアルゴリズム部分と上記ハードウェア構造の性能に影響を及ぼすアーキテクチャ部分とに分離した記述を保持する保持手段と、上記保持手段に保持したアルゴリズム部分の記述に基づいて演算を行うアルゴリズム演算手段と、上記保持手段に保持したアーキテクチャ部分の記述に基づいて演算を行うアーキテクチャ演算部とを有し、上記アルゴリズム演算手段とアーキテクチャ演算手段の演算により、上記ハードウェア構造の性能を評価することにより、上述した課題を解決する。

【００２２】すなわち、本発明の性能評価方法及び装置は、ＬＳＩ（大規模集積回路）アーキテクチャ設計の性能評価において、構造と機能を切り離してモードという単純なインターフェイスで関係を表現し、またアーキテクチャの記述を抽象化することにより、性能評価モデルを作成する場合に、最初から作成し直すのではなく、機能評価モデルをそのまま利用できるようにして性能評価のための設計工数を短くすることを可能とし、また、性能評価モデルの記述をより抽象的なレベルで行うようにしてシミュレーションの時間をより短縮することを可能としている。

【００２３】

【発明の実施の形態】本発明の好ましい実施の形態について、図面を参照しながら説明する。

【００２４】本発明の性能評価方法及び装置の具体的な実施の形態の説明に先立ち、例えば性能評価のための設計工数を短くし、また、シミュレーションの時間をより短縮するために本発明が採用している２つの方法と、本発明との比較のための一般的な性能評価モデルの構造と、本発明に係る性能評価モデルとについて、以下に説明する。

【００２５】本発明が採用している第１，第２の２つの方法のうち、第１の方法では、性能と機能を切り離して記述し、モードという抽象的なインターフェイスを定義してこの２つの関係を表現する。

【００２６】また、第２の方法では、上述の性能評価モデルにおけるアーキテクチャの記述を、リソーススケジューリングという考え方を用いて容易にし、シミュレーション実行環境を簡単に実現できるようにしている。

【００２７】ここで、上記第１の方法では、性能評価モデルの記述の中で、機能を表現しているアルゴリズム部分と、変化することで性能に影響を及ぼすアーキテクチャ部分とを切り離して記述する。また、この切り離しを実現するために、モードという抽象的なインターフェイスを定義して、アーキテクチャとアルゴリズムの関連を表現する。なお、以下では、このような時間的な概念を表すモデルをアーキテクチャと呼ぶようにする。

【００２８】図１には、本発明との比較のための一般的な性能評価モデルの構造を示す。なおここでは、設計対象のアーキテクチャの構成要素、つまりアーキテクチャの一部分を、ブロックと定義する。

【００２９】この図１において、あるデータが、ブロックＢＡを通過して、次に到達するブロックがブロックＢＢであるかブロックＢＣであるかは、ブロックＢＡでのデータの演算Ｃａの結果に依存する場合がある。ブロックＢＡ、ＢＢ、ＢＣでかかる処理時間がそれぞれＴＡ、ＴＢ、ＴＣであるとし、また、それらの処理時間ＴＡ、ＴＢ、ＴＣはそれぞれ常に一定であると仮定した場合、データに依存してこの系全体の処理にかかる時間は、（ＴＡ＋ＴＢ）である場合と（ＴＡ＋ＴＣ）である場合とがあり得る。この場合、データがブロックを通過する経路が機能に依存しており、そのためにデータの系全体の処理時間が機能に依存している。

【００３０】しかし、機能設計を行う際には、このようなアーキテクチャの構成を意識して行わないため、このようなモデルで性能評価を行うのならば、機能評価を行う時に作成した機能評価モデルは性能評価には全く利用できず、性能評価に当たっては性能評価モデルを最初から設計し直すこととなる。更に、機能自体は変化しないが、アーキテクチャ構造が変化した場合、一度作成した性能評価モデルを、構造が変化した部分に関して、機能も含めて作成し直す必要が生じる。しかし、機能には変化がないので本来であれば機能は作成し直す必要はないはずである。

【００３１】これに対して、図２で示される本発明の性能評価モデルでは、アルゴリズムとアーキテクチャが切り離されている。

【００３２】ここで、あるデータが演算Ｃａの処理を終了し、その結果次に演算Ｃｂが必要である、という場合をモードＭ１と定義する。また、あるデータが演算Ｃａの処理を終了し、その結果次に演算Ｃｃが必要である、という場合をモードＭ２と定義する。

【００３３】アーキテクチャ部では、モードＭ１の場合、データはブロックＢＡを通過してその次にブロックＢＢを通過する、というモデルを持っており、また、モードＭ２の場合、データはブロックＢＡを通過してその次にブロックＢＣを通過する、というモデルを持ってい

る。このモデルを利用することにより、アーキテクチャ部を、変化しないアルゴリズム部と切り離して、性能評価モデルの作成を行うことができる。したがって、性能評価のための設計工数を短縮することができることになる。

【００３４】このことにより、機能評価モデル決定後に例えばアーキテクチャ構造が変化した場合でも、一度作成した性能評価モデルは、機能に変化がなくアーキテクチャ部分とアルゴリズム部分のインターフェイスが変わらないならば、アーキテクチャ部分だけの変更で性能評価を行うことが可能となる。

【００３５】一方、ハードウェアアーキテクチャ構造は、パイプライン構造などの公知のアーキテクチャの例のように、機能によって変化しない。したがって、機能部分とアーキテクチャ部分の切り離された性能評価モデルを利用すれば、アーキテクチャの部品化を促進することができる。さらに、このアーキテクチャ構造の部品は機能を持たないため、上記「モード」のような部品のインターフェイスの統一が容易であり、そのため性能評価モデル作成の上でアーキテクチャ部品の再利用化が促進されることが期待される。

【００３６】次に、第２の方法では、上述の性能評価モデルにおけるハードウェアアーキテクチャの記述を、リソーススケジューリングという考え方を用いて容易にし、シミュレーション実行環境を簡単に実現できるようにしている。この結果、性能評価のための設計工数を短縮することが可能となる。

【００３７】また、アーキテクチャの記述をリソーススケジューリングという考え方を用いて抽象化したレベルで行うことで、シミュレーション実行時間を短縮することが可能となる。

【００３８】以下、リソーススケジューリングについての説明のために、トークンとリソースについて定義し、更にリソーススケジューリングについて定義する。

【００３９】トークンとは、ハードウェア内部で処理されるデータの任意の単位であると定義する。また、リソースとは、トークンを同一時間に限られた数のみ保持できる、ハードウェアアーキテクチャの構成要素であると定義する。さらに、リソーススケジューリングとは、設計者が決めることができない、種類に依存しない全てのリソースが持つ以下の３点の法則（ＵＡ１，ＵＡ２，ＵＡ３）にしたがって、リソースをスケジューリングすることである。なお、以下、法則のことをＵＡ（Universal Architecture）と呼ぶことにする。

【００４０】ＵＡ１では、最小単位のリソースの中にはトークンが同時に２つ以上存在することができない。

【００４１】ＵＡ２では、あるリソース中に存在するトークンはこのリソースから外に出ることが可能であり、次に移動する先のリソースが受付可能であるのならば必ず移動する。

【００４２】ＵＡ３では、どのようなリソースも、最小単位のリソースの組み合わせによって構成される。

【００４３】ここで、リソース間をトークンが移動するかどうか（以下ではこのことを遷移条件と呼ぶ）は、リソースの種類に応じて変化する。遷移条件はリソースの種類によって異なるポリシーの組み合わせによって決定される。ＵＡに加えて、リソースのポリシーが与えられることによって、リソースの種類（型）が決定する。

【００４４】以下、リソースのポリシーをＳＡ（Specific Architecture）と呼ぶことにする。

【００４５】実際のリソースは、上記リソースの種類に加えて、リソース内部の構造を定義したものになる。なお、リソースは階層的に定義可能であり、内部構造を持つリソースとは、内部にリソースを持つリソースのことである。内部にリソースを持たない最小単位のリソースには、内部構造はない。

【００４６】図３には、本発明におけるリソースの抽象化モデルの概要を示す。

【００４７】この図３において、「リソース」は、「リソースの種類の定義」及び「構造の定義」からなる。「リソースの種類の定義」は、どのリソースにも共通な「リソーススケジューリングの法則（ＵＡ）」と、種類によつて異なる「リソースのポリシー（ＳＡ）」からなる。

【００４８】リソースの「構造の定義」は、当該リソースが下位のどのようなリソースの組み合わせから構成されているかを示している。

【００４９】以下、リソースの構造について説明する。

【００５０】図４には、リソースの構造、つまりリソースの階層構造の例を示す。

【００５１】この図４において、リソースＡは、それよりも下位のリソースａ、リソースｂ、リソースｃの結合からなっている。なお、この図中の矢印は、トークンが遷移する方向を示している。

【００５２】図５には、モードＭ１，Ｍ２とリソースＡの結合との関係を示す。

【００５３】この図５において、同一リソースＡであっても、モードＭ１のときは、それよりも下位のリソースａ、リソースｂ、リソースｂの順にトークンが遷移するように結合されているが、モードＭ２のときは、下位のリソースａ，リソースｂの順にトークンが遷移するように結合されていると考える。つまり、リソースは、このような内部の構造をもっている。なお、上記モードＭ１の場合は、トークンが下位のリソースｂを二回繰り返して通過するように結合されている。

【００５４】また、リソースの結合は一対一の結合とは限らない。すなわち、図６に示すように、リソースａから、リソースｂ、ｃ、ｄに「分散」してトークンが遷移するような結合も考えられ、また、リソースｅ、ｄ、ｆからリソースｈに「集約」してトークンが遷移するよう

な結合も考えられる。

【００５５】本発明では、これらの定義を用いて、多くのハードウェアアーキテクチャが容易に記述できることを利用している。実際には、個々のハードウェアアーキテクチャは個々の特徴を持つが、以下に示すような限られた選択肢から選ぶことができる。

【００５６】ここで、あるリソースRがトークンを受け付けるポリシーを「ＳＡ１」とし、また、時刻tにリソースRにトークンが入ってきている状態が有りうることを、「時刻tにリソースRはトークンを受付可能である」、と定義する。

【００５７】このとき、あるリソースRがトークンを受け付けるポリシーとしては、例えば以下の３種類のポリシー（ＳＡ１－１、ＳＡ１－２、ＳＡ１－３）が考えられ、これらのうちの何れかが選択される。

【００５８】ＳＡ１－１では、あるリソースRが時刻tにおいてまさに空いているならば、時刻（ｔ＋１）にトークンを受け付けることが可能である。

【００５９】ＳＡ１－２では、あるリソースRの次にトークンが移動するであろうリソースR'が時刻（ｔ＋１）において受付可能であるとき、リソースRは時刻（ｔ＋１）に受付可能になる。

【００６０】ＳＡ１－３では、時刻tにおいてあるリソースRがまさに空いているか、又は次にトークンが移動するであろうリソースR'が時刻（ｔ＋１）において受付可能であるとき、リソースRは時刻（ｔ＋１）に受付可能である。なお、ＳＡ１－３とは、（ＳＡ１－１）＋（ＳＡ１－２）のポリシーである。

【００６１】また、トークンが分岐する場合、すなわちあるリソースRに存在するトークンがリソースRから外に出ることが可能で、複数のリソースＲ１，Ｒ２，・・・・，Ｒｎに同時に遷移する場合のポリシーを「ＳＡ２」とする。このときのポリシーとしては、例えば以下のＳＡ２－１が考えられる。

【００６２】ＳＡ２－１では、リソースRに存在するトークンが外に出ることが可能であり、リソースＲ１，Ｒ２，・・・・，Ｒｎが全て同時に受付可能である場合、トークンはリソースRからリソースＲ１，Ｒ２，・・・・，Ｒｎに分岐して遷移する。このとき、リソースＲ１，Ｒ２，・・・・，Ｒｎのそれぞれが受け付けることが可能であるかどうかは、ＳＡ１のいずれかのポリシー（ＳＡ１－１、ＳＡ１－２、ＳＡ１－３）に基づいて決定される。

【００６３】さらに、トークンが集約する場合、すなわち複数の入力元リソースＲ１，Ｒ２，・・・・，Ｒｎからトークンが１つのリソースRに遷移する場合のポリシーを「ＳＡ３」とする。このときのポリシーとしては、以下のＳＡ３－１が考えられる。

【００６４】ＳＡ３－１では、複数のリソースＲ１，Ｒ２，・・・，Ｒｎの全てにトークンが存在して、全てのトークンが外に出ることが可能であるのならば、トークンはリソースＲ１，Ｒ２，・・・・，ＲｎからリソースRに集約して遷移する。このとき、リソースRが受付可能であるかどうかは、ＳＡ１のいずれかのポリシー（ＳＡ１－１、ＳＡ１－２、ＳＡ１－３）に基づいて決定される。

【００６５】なお、本発明での具体的なポリシーの定義は、ＳＡ１については３種類（ＳＡ１－１、ＳＡ１－２、ＳＡ１－３）、ＳＡ２については１種類（ＳＡ２－１）、ＳＡ３については１種類（ＳＡ３－１）のみに止まっているが、これらは拡張可能であり、他のポリシーに基づいてモデル化することも考えられる。

【００６６】このようにして、ＵＡ（法則）と、個別のＳＡ（ポリシー）を予めライブラリとして提供することにより、ユーザすなわち性能評価者は、これらの組み合わせ及び構造を利用することによって簡易に性能評価モデルを作成することができ、性能評価のための設計工数を短縮することができる。

【００６７】図７には、本発明に係る性能評価モデルを作成・検証する一実施の形態の性能評価シミュレーション装置の概略構成例を示す。

【００６８】この図７の性能評価シミュレーション装置では、先ず、機能の設計の段階で作成した機能評価モデル６を、ユーザが拡張することによりモード計算部１を作成する。機能評価モデル６は、データの入力に対して演算結果を出力するが、モード計算部１は、演算結果ではなく、その演算結果に従ったモードを出力する。出力したモードは一時的にモードバッファ２に保存され、適切なタイミングで性能評価モデル部３に取り出され、利用される。

【００６９】性能評価モデル部３は、前述のようなリソースの抽象化に基いたモデルを持っていて、入力されたトークンを、適切な処理時間の後に出力する機能を果たしている。なお、本実施の形態の性能評価シミュレーション装置は、トークンの外部との制御のために入力可信号の出力や出力可信号の入力を持っている。入力可信号は、性能評価モデル部３が外部入出力管理部４からトークンを受付可能かどうかを制御する信号であり、出力可信号は、図示しない外部ブロックが性能評価モデル部３からトークンを受付可能かどうかを制御する信号である。性能評価モデル部３は、初期化時に、設計者の設計したアーキテクチャ構造について、前述のリソースの抽象化に基づいたモデルを実体化する。

【００７０】また、本実施の形態の性能評価シミュレーション装置は、モード計算部１や性能評価モデル部３と外部との入出力の間のインターフェイスとして、外部入出力管理部４を有している。

【００７１】さらに、本実施の形態の性能評価シミュレーション装置は、性能評価モデル部３や外部入出力管理部４に対して、ハードウェアクロックのタイミングを擬

似的に生成して、動作時間の単位を与えているクロック生成器５をも有している。

【００７２】以下、この図７に示した本実施の形態の性能評価シミュレーション装置の各部分の動作について説明する。

【００７３】先ず、モード計算部１の動作を説明する。図８には、モード計算部１の動作フローを示す。

【００７４】当該モード計算部１では、ステップＳ１にてデータが入力されると、ステップＳ２としてモードを計算し、さらにステップＳ３として当該計算されたモードをモードバッファ２に出力する。当該モード計算部１は、データが入力される毎にすぐに呼び出され、この図８で示されるフローの処理が行われる。

【００７５】次に、外部入出力管理部４の動作を説明する。図９及び図１０には、当該外部入出力管理部４の動作フローを示す。

【００７６】外部入出力管理部４では、クロック生成器５の生成する１クロック毎に処理を行う。先ず、外部入出力管理部４では、ステップＳ１１として、外部からの入力トークンがあるか否かの判断を行う。外部入出力管理部４は、このステップＳ１１の判断において、外部からの入力トークンがあると判断した場合（ＹＥＳ）にはステップＳ１２の処理に進み、当該ステップＳ１２にてデータをモード計算部１に渡した後、ステップＳ１３の処理に進む。一方、ステップＳ１１にて外部からの入力トークンが無いと判断した場合（ＮＯ）、外部入出力管理部４の処理はステップＳ１３に進む。

【００７７】外部入出力管理部４は、ステップＳ１３の処理に進むと、モードバッファ２の内容を参照し、当該モードバッファ２にモード情報が存在し、且つ性能評価モデル部３からの入力可信号が入力可であるか否かの判断を行う。外部入出力管理部４は、当該ステップＳ１３の判断において、モードバッファ２にモード情報が存在し、且つ性能評価モデル部３からの入力可信号が入力可であると判断した場合（ＹＥＳ）にはステップＳ１４の処理に進み、当該ステップＳ１４にて性能評価モデル部３に対して、外部から入力されていたトークンを渡す。一方、ステップＳ１３にてモードバッファ２にモード情報が存在しないか、或いは、性能評価モデル部３からの入力可信号が入力可で無いと判断した場合（ＮＯ）、外部入出力管理部４の処理は、ステップＳ１５に進む。

【００７８】外部入出力管理部４は、ステップＳ１５の処理に進むと、性能評価モデル部３からトークンが出てきており、且つ外部へ出力可能であるか否かの判断を行う。外部入出力管理部４は、当該ステップＳ１５の判断において、性能評価モデル部３からトークンが出てきており、且つ外部へ出力可能であると判断した場合（ＹＥＳ）にはステップＳ１６の処理に進み、当該ステップＳ１６にてそのトークンを外部に出力し、その後、図１０のステップＳ１７の処理に進む。一方、ステップＳ１５にて性能評価モデル部３からトークンが出力されないか、或いは、外部へ出力できないと判断した場合（ＮＯ）、当該外部入出力管理部４の処理は、図１０のステップＳ１７の処理に進む。

【００７９】外部入出力管理部４は、ステップＳ１７の処理に進むと、モードバッファ２を参照し、次クロックでモードバッファ２の内容が存在するか否かの判断を行う。外部入出力管理部４は、当該ステップＳ１７の判断において、次クロックでモードバッファ２の内容が存在すると判断した場合（ＹＥＳ）にはステップＳ１８の処理に進み、当該ステップＳ１８にて外部からの入力可信号を入力不可とする。一方、ステップＳ１７にて次クロックでモードバッファ２の内容が存在しないと判断した場合（ＮＯ）、外部入出力管理部４では、ステップＳ１９として、外部からの入力可信号を入力可とする。なお、外部入出力管理部４では、ステップＳ１７において、モードバッファ２の中にモード情報が存在しないか、あるいはモードバッファ２の中にモード情報が１つ存在し且つステップＳ１４の処理を行ったか、のいずれかの場合に、次クロックでモードバッファ２の内容は存在しないと判断する。ステップＳ１８、ステップＳ１９の処理後、外部入出力管理部４の処理はステップＳ２０に進む。

【００８０】ステップＳ２０の処理に進むと、外部入出力管理部４は、外部から与えられた出力可信号を性能評価モデル部３への出力可信号に反映する。

【００８１】次に、性能評価モデル部３の動作を説明する。図１１には、性能評価モデル部３の動作フローを示す。

【００８２】当該性能評価モデル部３でも、上記外部入出力管理部４と同様にクロック生成器５の生成する１クロック毎に処理を行う。

【００８３】性能評価モデル部３では、ステップＳ２１として、外部入出力管理部４からの入力トークンが有ったか否かの判断を行う。性能評価モデル部３は、ステップＳ２１の判断において、外部入出力管理部４からの入力トークンが有ったと判断した場合（ＹＥＳ）にステップＳ２２の処理に進み、当該ステップＳ２２にてモード計算部１によって計算されたモードをモードバッファ２から取り出し、さらにステップＳ２３にて外部からの入力端になるリソースに後述するトークン入力動作を行う。このステップＳ２３の処理後、性能評価モデル部３の処理はステップＳ２４に進む。一方、ステップＳ２１の判断において、外部入出力管理部４からの入力トークンが無いと判断した場合、性能評価モデル部３はステップＳ２４の処理に進む。

【００８４】ステップＳ２４の処理に進むと、性能評価モデル部３では、展開された全ての最小リソース間について、存在する全てのトークンの移動を行う。次いで、性能評価モデル部３では、ステップＳ２５として、外部

への出力端となるリソースについて、外部入出力管理部４のと間のトークン出力動作を行う。なお、「展開された最小リソース」の意味については後述する。

【００８５】ここで、図１２には性能評価モデル部３の構成例を示す。図１１のステップＳ２３，Ｓ２４，Ｓ２５の処理は、当該図１２に示す構成により行われる。

【００８６】この図１２において、性能評価モデル部３は、構造定義部１１と、複数のリソース１２と、それらリソース１２を保持するリソース保持部１３、ＵＡ保持部１４、ＳＡ保持部１５及びリソース内のトークンの動作を制御するトークン制御部１６からなっている。

【００８７】構造定義部１１には、ユーザであるところのＬＳＩ設計者によって設計対象のアーキテクチャ構造が記述される。この記述はアーキテクチャ構造を構成する各リソースの種類と、それらのリソース間の階層構造が記述されている。ただし、この階層構造は、モードに依存して変化するものであり、そのためにモード毎に別々の階層構造が記述されている。図１３には、ユーザの記述する構造定義部１１の記述例を示す。

【００８８】また、性能評価モデル部３は、初期化処理時に、構造定義部１１の記述に基づいて、最小リソース保持部１３内にリソースのモデルを実体化する。この際には、前記ＵＡ３すなわち「どのようなリソースも、最小単位のリソースの組み合わせによって構成される」ことに基づいて、構造定義部１１で記述されたリソースの階層構造を最小単位のリソースに展開した形でモデルを実体化する。最小単位のリソースの中には、トークンは同時に２つ以上存在することができない（前記ＵＡ１）ことから、最小単位のリソースに展開した形でモデルを実体化することにより、トークン制御部１６によるトークンの制御を単純な形にしている。図１４には、最小単位のリソースへの展開の概念図を示す。すなわち、性能評価モデル部３は、リソースａ，ｂ，ｃからなるリソースＡとリソースｊ，ｋからなるリソースＢを最小単位のリソースａ，ｂ，ｃ，ｊ，ｋに展開する。

【００８９】トークン制御部１６は、外部より入力されるトークン、モード、出力可信号にしたがって、ＵＡ保持部１４とＳＡ保持部１５にそれぞれ保持されたＵＡ，ＳＡに基づいて、最小リソース保持部１３に保持されたリソースのモデル中に存在するトークンの制御を行っている。つまり、性能評価モデル部３の動作フローである前記図１１に示された一連の処理は、トークン制御部１６が主体となって処理している。

【００９０】以下、図１２に示した構成の性能評価モデル部３を参照しながら、図１１で示されたステップＳ２３、Ｓ２４、Ｓ２５の説明を行う。なお、以下の説明では、簡単のために、トークン制御部１６の処理の基本となるステップＳ２４について最初に説明し、その後ステップＳ２３、Ｓ２５についてそれぞれ説明することにする。

【００９１】図１５には、リソース間のトークン移動の基本動作を示す。すなわち性能評価モデル部３の動作フローである前記図１１のステップＳ２４である、全ての最小リソース間のトークン移動動作は、図１５で示されたリソース間のトークン移動の基本動作を、全ての結合された最小リソース間に渡って実行することにより実現される。

【００９２】先ず、トークン制御部１６では、ステップＳ３１及びＳ３２の処理として、結合された任意の２つの最小リソースＲ１，Ｒ２について、リソースＲ１の出力条件を判断する。ここで、トークン制御部１６は、ステップＳ３１及びＳ３２の判断処理において、リソースＲ１の出力条件が満たされたと判断した場合（ステップＳ３２がＹＥＳ）、ステップＳ３３の処理に進む。一方、リソースＲ１の出力条件が満たされないと判断した場合（ステップＳ３２がＮＯ）、リソース制御部１６の処理は終了する。

【００９３】トークン制御部１６は、ステップＳ３３の処理に進むと、ステップＳ３１及びＳ３３の処理として、結合された任意の２つの最小リソースＲ１，Ｒ２について、リソースＲ２の入力条件を判断する。ここで、トークン制御部１６は、ステップＳ３３の判断処理において、リソースＲ２の出力条件が満たされたと判断した場合（ＹＥＳ）、ステップＳ３４としてトークンをリソースＲ１からＲ２に移動する。一方、ステップＳ３３の判断において、リソースＲ２の出力条件が満たされないと判断した場合（ＮＯ）、リソース制御部１６の処理は終了する。

【００９４】以下、上述のように２つのリソース間の、１つのトークンの移動に着目した場合、リソースＲ１をソースリソース、リソースＲ２をデスティネーションリソースと呼ぶことにする。

【００９５】更に、図１６には、図１５のステップＳ３２における出力条件判断動作の詳細を示す。

【００９６】この図１６において、トークン制御部１６は、ステップＳ４１としてソースリソース中にトークンが１つ存在しているかどうかを判断する。

【００９７】このステップＳ４１の判断において、ソースリソース中にトークンが１つ存在していないと判断した場合（ＮＯ）、トークン制御部１６は、ステップＳ４４として、前記ＵＡ２により、このソースリソースはトークンを出力することができないとする。一方、ステップＳ４１の判断において、ソースリソース中にトークンが１つ存在していると判断した場合（ＹＥＳ）、トークン制御部１６は、さらにステップＳ４２として、出力先のリソースはデスティネーションリソース以外にも存在するか否か、つまり出力先のリソースは複数かどうかを判断する。

【００９８】このステップＳ４２の判断により、出力先のリソースは複数でないと判断した場合（ＮＯ）、トー

クン制御部16は、ステップS45として、トークンを出力可とする。また、ステップS42の判断において、出力先のリソースは複数であると判断した場合（YES）、トークン制御部16は、更にステップS43として、ソースリソースの前記SA2（分岐）の条件が満たされるかどうかを判断する。

【0099】このステップS43の判断において、ソースリソースのSA2（分岐）の条件が満たされないと判断した場合（NO）、トークン制御部16は、ステップS44の処理に進み、このソースリソースはトークンを出力することができないとする。一方、ステップS43の判断において、ソースリソースのSA2（分岐）の条件が満たされたと判断した場合（YES）、トークン制御部16は、ステップS45として、トークンを出力可とする。なお、SA2の条件はそのソースリソースによって異なるが、前記SA2-1をとるソースリソースの場合には、出力先のリソースのすべてについて、それぞれが入力可能であるかどうかを判断することになる。次に、図17には、図15のステップS32における入力条件判断動作の詳細を示す。

【0100】この図17において、トークン制御部16は、ステップS51としてデスティネーションリソース中にトークンが存在していて、且つ、デスティネーションリソースの更に次にトークンが移動するであろうリソースからデスティネーションリソースへの入力可信号が入力不可であるかどうかの条件判断を行う。

【0101】このステップS51の判断において、上記条件が満たされる場合（YES）、トークン制御部16は、ステップS55としてトークンは入力不可とする。一方、ステップS51の判断において、上記条件が満たされない場合（NO）、トークン制御部16は、続いてステップS52として、デスティネーションリソースの前記SA1（受付）の条件が満たされるかどうかを判断する。

【0102】このステップS52の判断において、上記SA1の条件が満たされないと判断した場合（NO）、トークン制御部16は、ステップS55としてトークンは入力不可とする。また、ステップS52の判断において、上記SA1の条件が満たされると判断した場合、トークン制御部16は、更にステップS53として、デスティネーションリソースの入力元のリソースはソースリソース以外にも存在する、つまり、入力元のリソースは複数であるかどうかを判断する。

【0103】このステップS53の判断において、入力元のリソースは複数でないと判断した場合（NO）、トークン制御部16はステップS56としてトークンは入力可とする。一方、ステップS53の判断において、入力元のリソースは複数であると判断した場合（YES）、トークン制御部16は、更にステップS54として、デスティネーションリソースの前記SA3（集約）の条件が満たされるかどうかを判断する。なお、SA3の条件はそのデスティネーションリソースによって異なるが、SA3-1をとる場合は、入力元のリソースのすべてについて、それぞれがトークンを出力可能であるかどうかを判断することになる。

【0104】このステップS54の判断において、前記SA3の条件が満たされると判断した場合（YES）、トークン制御部16は、ステップS56としてトークンは入力可とする。一方、このステップS54の判断において、前記SA3の条件が満たされないと判断した場合（NO）、トークン制御部16は、ステップS55としてり、トークンは入力不可とする。

【0105】次に、前記図11で示したように、最小リソース間のトークンの移動とは別に、外部からの入力端になるリソースへのトークンの入力と、外部への出力端となるリソースからのトークンの出力動作があり、この動作について以下に説明する。

【0106】図18には、図11のステップS23である入力端リソースのトークン入力動作を示す。

【0107】ここで、トークンが通る入力端と出力端のリソースをそれぞれ1づつ持つリソースの一連の結合を、リソースのパスと定義する。

【0108】トークン制御部16は、先ずステップS61及びS62として、各モードに対するパスの先頭リソースについて、リソースのUA1とSA1の条件が満たされるかどうかを判断する。なお、UA1の条件とは、前述の図17におけるステップS51の条件と同じである。このステップS61及びS62の判断において、UA1とSA1の条件が満たされないと判断した場合（NO）、トークン制御部16は、ステップS63として外部入出力管理部4に入力不可通知を出して処理を終了する。一方、ステップS61及びS62の判断において、UA1とSA1の条件が満たされると判断した場合（YES）、トークン制御部16は、ステップS63として、モードバッファ2からモード情報を取り出し、さらに性能評価モデル部3の現在のモードを今取り出したモードに変更する。

【0109】その後、トークン制御部16は、ステップS65として、外部入出力管理部4からのトークンを、入力端リソースに移動する。

【0110】次に、図19には、図11のステップS25である出力端リソースのトークン出力動作を示す。

【0111】ここで、リソースのパスはモードによって異なるため、モード毎のリソースパスの終端リソースは複数存在し得る。しかし、それらの終端リソースの次に全てのモードにおいて存在する仮想的な出力端リソースを仮定する。

【0112】この場合の仮想出力端リソースへのトークンの遷移条件は以下のようになる。

【0113】各モード毎の終端リソースが、互いに異な

るリソースR1，R2，・・・，Rnであり、これらのいずれか2つ以上に同時にトークンが存在して、それらのトークンが外に出ることが可能であるのならば、最も発生時期の古いトークンだけが仮想出力端リソースRに遷移する。このとき、リソースRが受付可能であるかどうかは、外部入出力管理部4からの出力可信号に基づいて決定される。

【0114】この条件のもとで、トークン制御部16は、先ずステップS71及びS72として、各モード毎のパスを統合する仮想出力端リソースについて、リソースのトークン出力条件が満たされるかどうかを判断する。

【0115】当該ステップS71及びS72の判断において、リソースのトークン出力条件が満たされないと判断した場合（NO）、トークン制御部16は処理を終了する。一方、ステップS71及びS72の判断において、リソースのトークン出力条件が満たされると判断した場合（YES）、トークン制御部16は、ステップS73として、外部入出力管理部4からの出力可信号があるかどうかを判断する。

【0116】このステップS73の判断において、外部入出力管理部4からの出力可信号がないと判断した場合（NO）トークン制御部16は処理を終了する。一方、ステップS73の判断において、外部入出力管理部4からの出力可信号があると判断した場合（YES）、トークン制御部16は、ステップS74として、外部入出力管理部4にトークンを出力する。

【0117】以上のように、本発明実施の形態によれば、機能を表現しているアルゴリズム部分と性能に影響を及ぼすアーキテクチャ部分とが切り離され、モードというインターフェイスでその2つの関係を記述した性能評価モデル作成方法及びそのモデルの性能評価がシミュレーションできる性能評価シミュレーション装置を用いることによって、性能評価モデルの作成時に機能評価モデルで作成したアルゴリズム部の再利用が可能となり、性能評価のための設計工数を従来よりも短縮することができる。

【0118】また、本発明実施の形態によれば、同様にして機能評価モデルと独立に性能評価モデルの作成が可能となるので、機能評価モデルが完成しないうちに性能評価モデルの作成が開始できるため、性能評価のための設計工数を従来よりも短縮することができる。

【0119】更に、本発明実施の形態によれば、同様に、機能に依存しない性能評価モデルの作成が可能であることに加え、本発明における性能評価モデルのアーキテクチャ記述方法及びそのモデルの性能評価シミュレーションができる性能評価シミュレーション装置を用いることによって、階層的なアーキテクチャの記述が可能となるので、一般的なハードウェアアーキテクチャ構造を部品化して一旦作成したモデルを再利用することができ、その結果、性能評価のための設計工数を従来よりも短縮することができる。

【0120】また、本発明実施の形態においては、リソーススケジューリングという考え方を用いることによって、性能評価モデルのアーキテクチャ記述方法が容易となり、性能評価のための設計工数を従来よりも短縮することができる。

【0121】更に、本発明実施の形態においては、リソーススケジューリングという考え方に基づいたアーキテクチャ構造の抽象化を用いることにより、そのモデルの性能評価シミュレーションができる性能評価シミュレーション装置の実行を抽象化されたレベルで行うことができるため、シミュレーション実行時間を従来よりも短縮することができる。

【0122】

【発明の効果】以上の説明で明らかなように、本発明の性能評価方法及び装置においては、ハードウェア構造の性能を評価するための性能評価モデルに関する記述を、機能を表現しているアルゴリズム部分と性能に影響を及ぼすアーキテクチャ部分とに分離して記述することにより、性能評価のための設計工数を短くでき、また、シミュレーションの時間をより短縮することを可能としている。

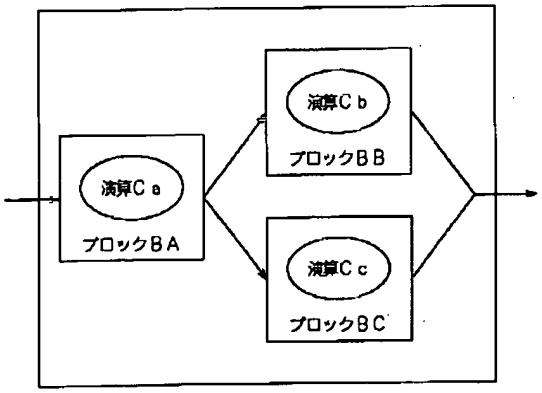
【0123】すなわち、本発明によれば、性能評価モデルの作成時に機能評価モデルで作成したアルゴリズム部の再利用が可能となり、性能評価のための設計工数を従来よりも短縮することができる。また、本発明によれば、機能評価モデルと独立に性能評価モデルの作成が可能となるので、機能評価モデルが完成しないうちに性能評価モデルの作成が開始できるため、性能評価のための設計工数を従来よりも短縮することができる。更に、本発明によれば、機能に依存しない性能評価モデルの作成が可能であることに加え、階層的なアーキテクチャの記述が可能となるので、一般的なハードウェアアーキテクチャ構造を部品化して一旦作成したモデルを再利用することができ、その結果、性能評価のための設計工数を従来よりも短縮することができる。また、本発明においては、リソーススケジューリングという考え方を用いることによって、性能評価モデルのアーキテクチャ記述方法が容易となり、性能評価のための設計工数を従来よりも短縮することができる。更に、本発明においては、リソーススケジューリングという考え方に基づいたアーキテクチャ構造の抽象化を用いることにより、そのモデルの性能評価シミュレーションができる性能評価シミュレーション装置の実行を抽象化されたレベルで行うことができるため、シミュレーション実行時間を従来よりも短縮することができる。

【図面の簡単な説明】

【図1】一般的な性能評価モデルの構造の説明に用いる図である。

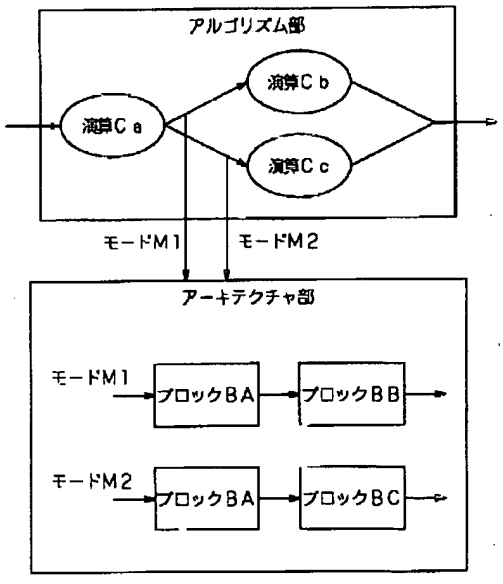
【図２】本発明の性能評価モデルの構造の説明に用いる図である。
【図３】本発明におけるリソースの抽象化モデルの概要説明に用いる図である。
【図４】リソースの階層構造の一例を示す図である。
【図５】モードとリソースの結合との関係の説明に用いる図である。
【図６】リソース間トークンの分散と集約の説明に用いる図である。
【図７】本発明に係る性能評価モデルを作成・検証する一実施の形態の性能評価シミュレーション装置の概略構成例を示すブロック図である。
【図８】モード計算部の動作を示すフローチャートである。
【図９】外部入出力管理部の１クロック毎の動作（その１）を示すフローチャートである。
【図１０】外部入出力管理部の１クロック毎の動作（その２）を示すフローチャートである。
【図１１】性能評価モデル部の１クロック毎の動作を示すフローチャートである。
【図１２】性能評価モデル部の構成例を示すブロック図である。
【図１３】ユーザの記述する構造定義部の記述例を示す図である。
【図１４】最小単位のリソースへの展開の概念図である。
【図１５】リソース間のトークン移動の基本動作を示すフローチャートである。
【図１６】図１５のステップＳ３２における出力条件判断動作（ソースリソースのトークン出力条件判断動作）の詳細を示すフローチャートである。
【図１７】図１５のステップＳ３２における入力条件判断動作（リソースのトークン入力条件判断動作）の詳細を示すフローチャートである。
【図１８】図１１のステップＳ２３である入力端リソースのトークン入力動作のフローチャートである。
【図１９】図１１のステップＳ２５である出力端リソースのトークン出力動作を示すフローチャートである。
【図２０】ＬＳＩ設計者がＬＳＩを設計する場合の従来の設計動作のフローチャートである。
【符号の説明】
１　モード計算部、　２　モードバッファ、　３　性能評価モデル、　４　外部入出力管理部、　５　クロック生成器、　６　機能評価モデル、　１１　構造定義部、　１２　リソース、　１３　最小リソース保持部、　１４　ＵＡ保持部、　１５　ＳＡ保持部、　１６　トークン制御部

【図１】



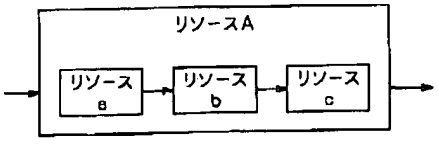
一般的な性能評価モデルの構造

【図２】



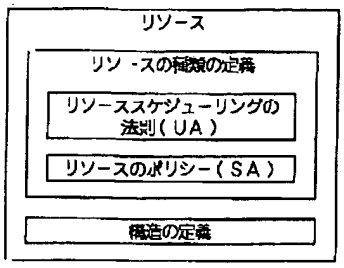
本発明の性能評価モデルの構造

【図４】



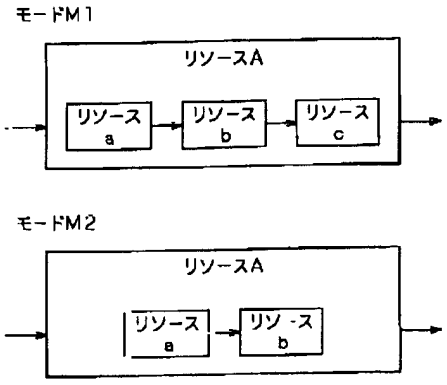
リソースの構成

【図3】



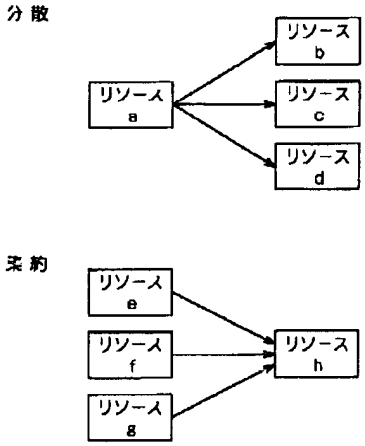
リソースの抽象化モデル

【図5】



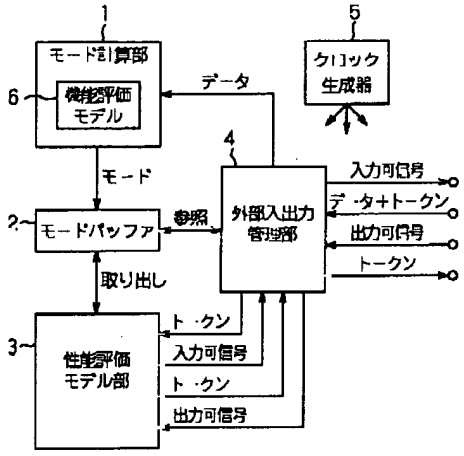
モードとリソースの結合の関係

【図6】



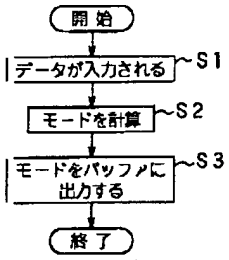
リソース間トークンの分散と集約

【図7】



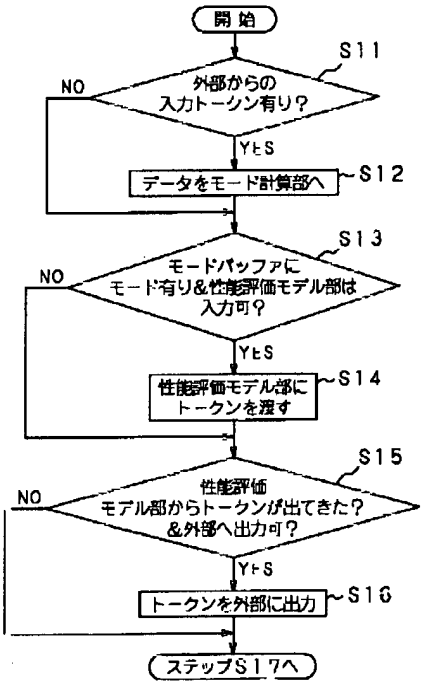
性能評価シミュレーション装置構成

【図8】



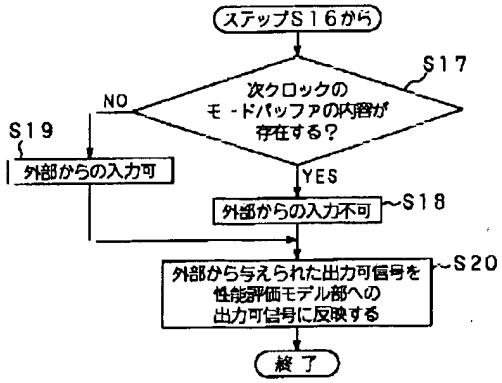
モード計算部動作

【図9】



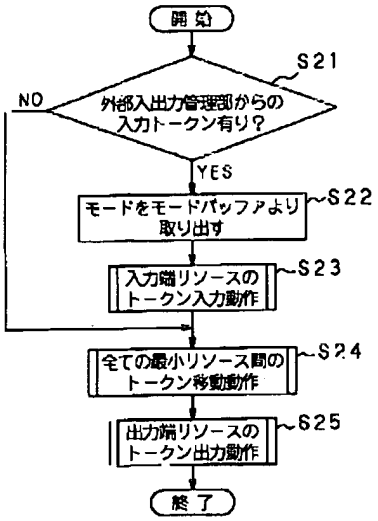
外部入出力管理部の1クロックごとの動作
(その1)

【図10】



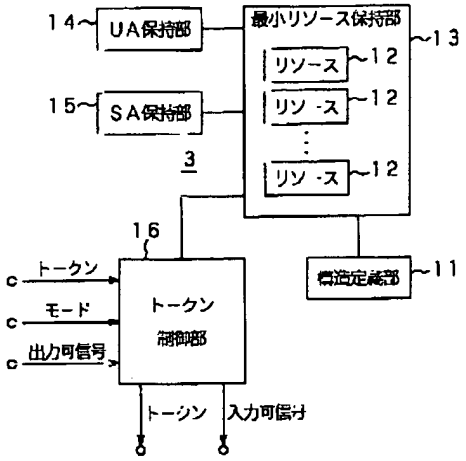
外部入出力管理部の1クロックごとの動作
(その2)

【図11】



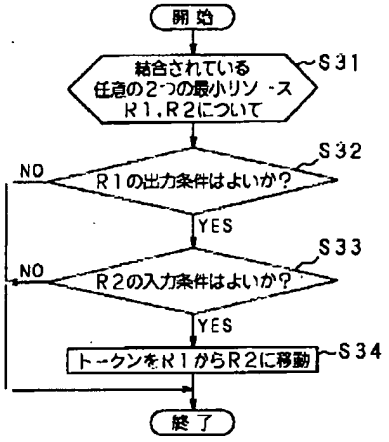
性能評価モデル部の1クロックごとの動作

【図12】



性能評価モデル部構成

【図15】



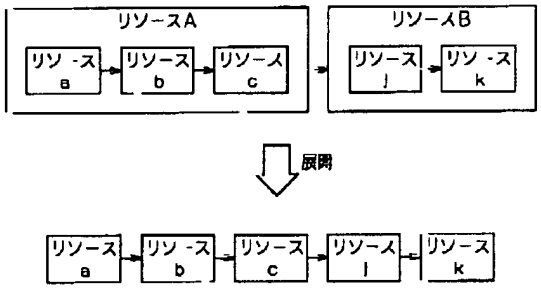
リソース間のトークン移動の基本動作

【図13】

```
//リソース作成
NewSimplePipeline,A;
  //SimplePipeline型のリソース名'A'として
    実体化する

NewSimplePipeline,B;
NewCustomPipeline,C;
NewRegister,bR1;
NewRegister,bR2;
NewRegister,cR1;

//モード毎のデータパス登録
RegPath B,1,bR1+bR2;
  //リソース'B'では、モード'1'のときに
    トークンはあらかじめ宣言されたリソース
    'bR1','bR2'の順に流れる

RegPath B,2,bR1+bR1+bR2;
RegPath B,3,bR1+bR1+bR2+bR2;

RegPath C,1,cR1+cR1;
RegPath C,2,cR1;
RegPath C,3,cR1+cR1+cR1;

RegPath A,1,B+C;
RegPath A,2,B+C;
RegPath A,3,B+C;

//最上位リソース登録
RegTop A;
  //この性能評価モデルのうち、外部入出力管理部と
    直接やり取りをするのはリソース'A'である
```

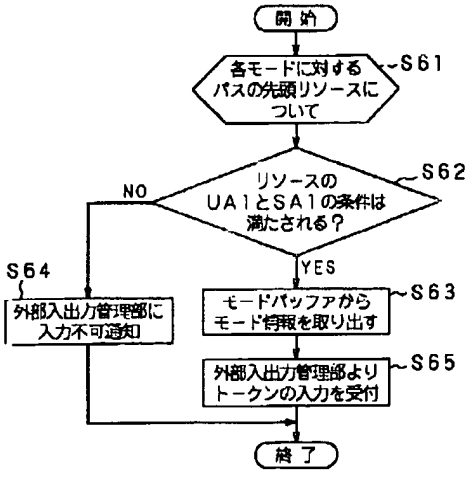
構造定義部の記述の例

【図14】



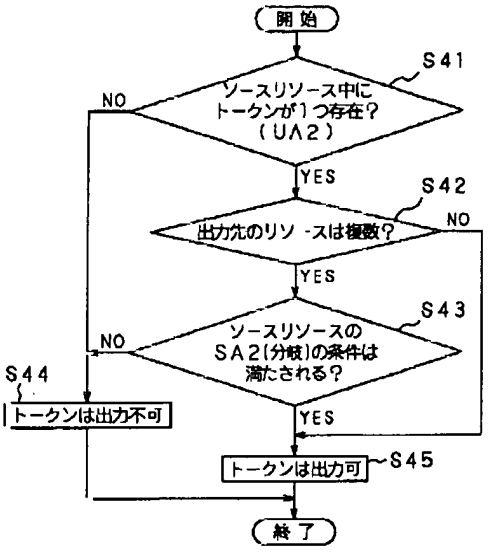
最小単位のリソースへの展開

【図18】



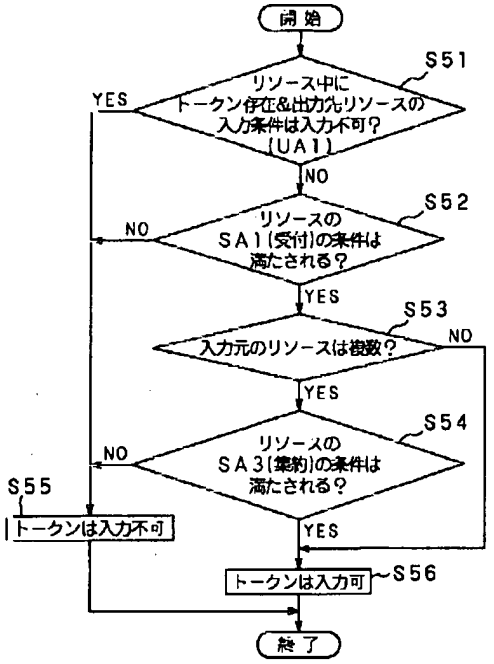
入力端リソースのトークン入力動作

【図16】



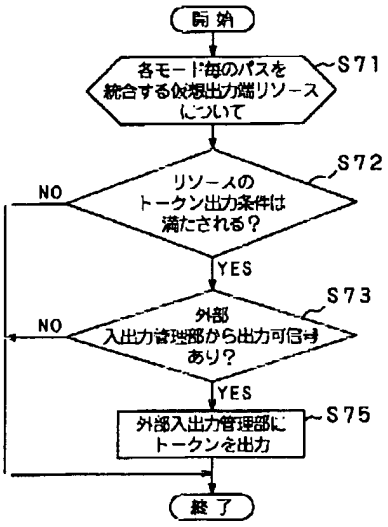
ソースリソースのトークン出力条件判断動作

【図17】



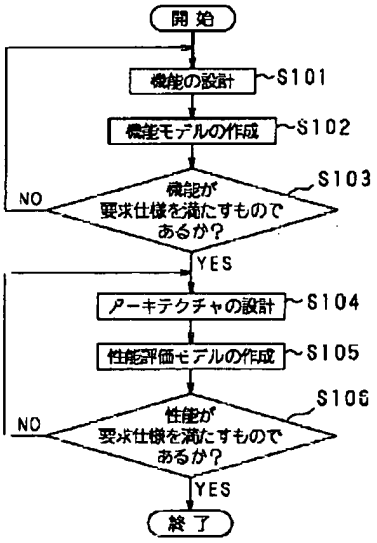
リソースのトークン入力条件判断動作

【図19】



出力端リソースのトークン出力動作

【図20】



従来のＬＳＩ設計フロー